Regular Expressions

Formele en Natuurlijke Talen

Lecture 5

Roadmap for today

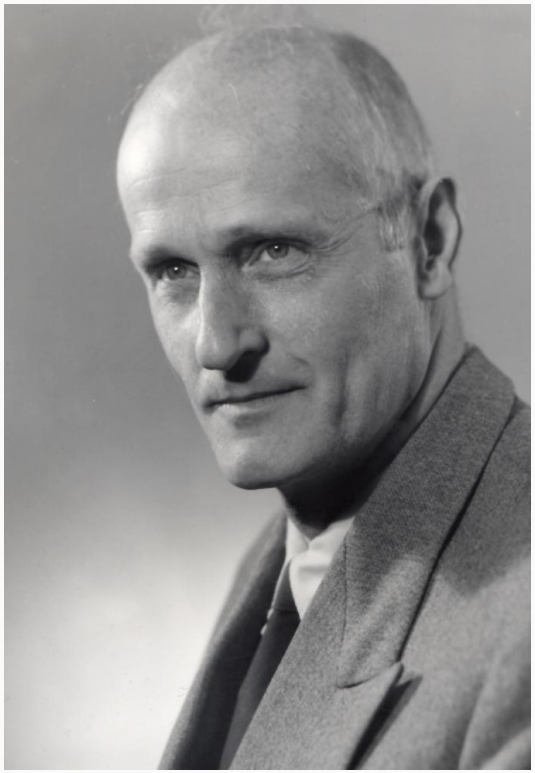
- What are regular expressions?
- Why are they useful?
- Basic patterns & exercises
- Relating regular expressions and FSAs

Crash Course in Regular Expressions

What is a regular expression ('regex')?

/xy*a.+[5-9z](x|y2k.)[^\dabc]*\w*$/

- Algebraic notation, characterizes a set of strings (=a **language**)
- Used to find strings which *match* certain patterns
- Our convention: occur between slashes/ /, which are **not** themselves part of the expression
- Brainchild of Kleene (1951)



Practical applications

Very useful in text processing and search:

- Finding things with conventional formats (prices, URLs, filenames…)
- Extracting patterns from **corpora** (bodies of text)
- Part of the standard library of most programming languages

Buildings Blocks of REs

Syntax of regular expressions over alphabet Σ:

1. $\emptyset$ and ϵ are regular expressions
2. For all x in Σ, `/x/` is a regular expression
3. If `/A/` and `/B/` are regular expressions, then

    (a) `/AB/` is a regular expression (**concatenation**)
    (b) `/A|B/`is a regular expression (**alternation**/**union**)
    (c) `/A*/` is a regular expression (**repetition**/**Kleene star**)

Notation: For regex R, L(R) = the set of strings which match R

Interpreting regular expressions

The basic operations have familiar interpretations:

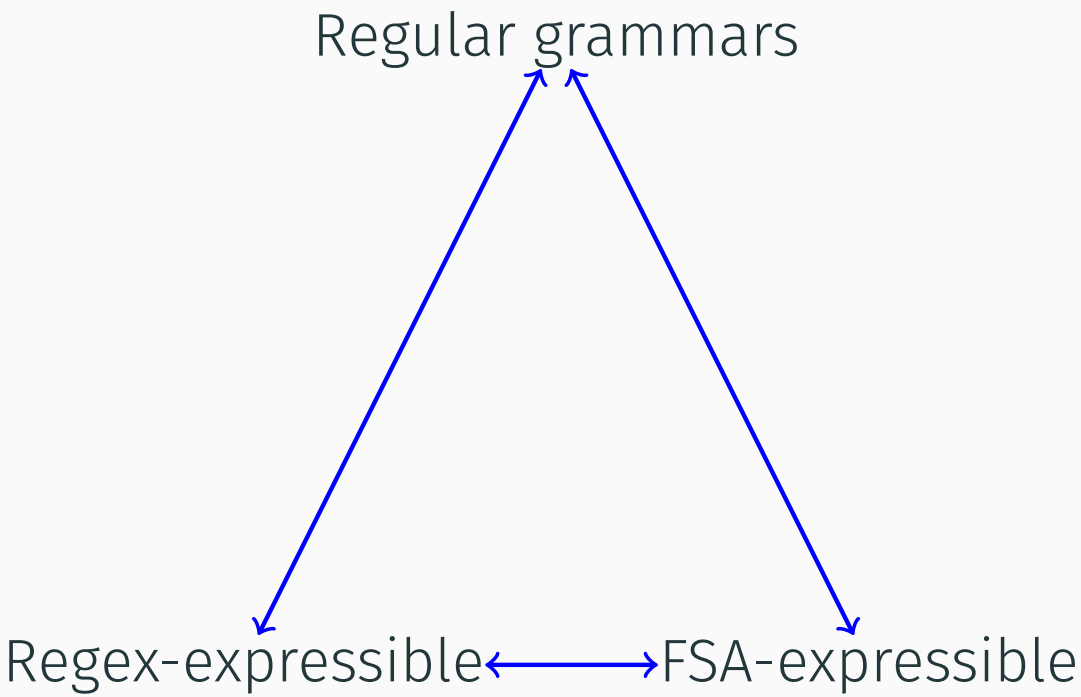
RegEx	Description	Example matches
`/x/`	the string 'x'	x
`/xy/`	concatenation of x and y	xy
`/x|y/`	either x or y	x, y
`/x*/`	zero or more x	ϵ, x, xx, xxx, ...

Concatenation is unmarked, so `/egel/` matches only exactly *egel*

Wildcard character: /./ matches **any single** character

Expressivity of regexes

What kind of languages can be represented with regexes?



Compiled regular expressions

What makes regexes especially useful is extensive **shortcuts** for common functions.

- ⋆ All functions are defined in terms of concatenation, alternation, and the Kleene star

🚨 BEWARE! 🚨 Actual regex compilers may vary in notation or even incorporate operators which are not **formally** regular!

Some common shortcuts: Quantifiers

Quantifiers: tell you how many times you can repeat the preceding character (like *)

RegEx	Description	Example matches
/x+/	at least one x (=/xx*/	x, xx, xxx, ...
/x?/	at most one x (=/x|ϵ/)	ϵ, x
/(xy)*/	at least zero xy	ϵ, xy, xyxy, xyxyxy, ...
/xy*/	x concenated with at least zero y	x, xy, xyy, xyyy, ...

Precedence

Order of operations matters!

```
/dog|cat/
```

If concatenation precedes |: matches *dog* and *cat*

If | **precedes concatenation**: matches *docat* and *dogat*

We want regular expressions to be unambiguous, so we need to make an assumption one way or the other.

Precedence

Like arithmetic, a **strict hierarchy** of operations:

Parentheses	>	*Quantifiers*	>	*Sequences*	>	*Alternation*
`( )`		`* + ?`		`meow`		`|`

What does /ca*t|do?g/ match?

Practice

1 Ga naar **wooclap.com**

2 Voer de code van het evenement in de bovenste banner in

Evenementcode
FENT05

Antwoorden per sms inschakelen

Practice

Which of the following strings is matched by the regular expression $(ab^*a)^*ab^*(d|cb^*)$?

(1) a ❌

(2) abbbaabdb ❌

(3) aaaacbbb ❌

(4) ac 👍

Practice

What simplest regular expression would match only...

1. All strings consisting of an even number of a's followed by an odd number of b's? /(aa)*b(bb)*/
2. All strings starting with 'wom' and ending with 'bat´? /wom.*bat/
3. The US vs. UK spellings of *traveler*/*traveller*? /travell?er/
4. The names *Maarten* and *Martijn*? /Ma(arte|rtij)n/
   Why not /Ma?rt(e|ij)n/?

Character classes

We might sometimes care about whether a substring is in a particular set (like letters or digits)

Character classes: Match any single character between square brackets [ ]. Equivalent to iterative alternation.

RegEx	Description	Examples
`[qxj7]`	any of q, x, j, 7 (= `q|x|j|7`)	q, x, j, 7

A carat ^ at the beginning of a class gives you its **complement**: it matches everything *except* members of that class.

RegEx	Description	Examples
`[^qxj7]`	any char. besides q,x,j,7	a, 2, !, #, ...

Character classes: ranges

Some important classes, like letters and numbers, are conventionally ordered, so we can make use of **ranges**:

RegEx	Description	Examples
`[a-z]`	any lowercase letter	a, b, c, ..., z
`[C-Q]`	any uppercase letter from C to Q	C, D, E, ..., Q
`[3-9]`	any digit from 3 to 9	3, 4, 5, ..., 9
`[a-zA-Z]`	any lowercase/uppercase letter	a, A, b, ..., Z

Character classes: aliases

Certain special character classes also have abbreviations:

RegEx	Expansion	Description	Examples
`\d`	`[0-9]`	any digit	0, 1, 2,...
`\D`	`[^0-9]`	any non-digit	a,b,!,...
`\w`	`[A-Za-z0-9_]`	any alphanumeric	a, 0, 1, _, ...
`\W`	`[^A-Za-z0-9_]`	any non-alphanumeric	!, , ?
`\s`		any whitespace char.	space, tab

NB: 'Alphanumeric' includes underscores _

Anchors

Important text-specific properties: beginning and ending words and lines

RegEx	Match
^	beginning of line
$	end of line
\b	word boundary
\B	non-word boundary

'word' = string of numbers, letters, underscores

NB! ^ means complementation only at the beginning of a character class.

Escape characters

To match characters that have special meanings in REs, precede them with \:

RegEx	Description	Example strings
`\.`	period	.
`\?`	question mark	?
`\+`	plus sign	+
`\\`	backslash	\

Practice

Come up with the simplest regular expression to match only...

1. Any email address at a *.com* or *.co.uk* domain
   e.g. `\w[\w\.]*@\w[\w\.]*\.co(m|\.uk)`
2. The words *wombat, wombats, wombatje, wombatjes* as they might appear in a corpus of texts
   e.g. `/^[Ww]ombat(je)s?(\b|[\.,\?\!;"':\(\)]*)/`

(There might be multiple right answers depending on the assumptions you make!)

Regular Expressions vs. Finite State Automata

REs vs. FSAs

Regular expressions and FSAs describe the same set of languages, i.e. the regular ones.

Regex	FSA
Represented as string	Represented as graph
Compact	Detailed
Machine-readable	Human-readable

Because of their equivalent expressive power, we can **convert between FSAs and REs**.

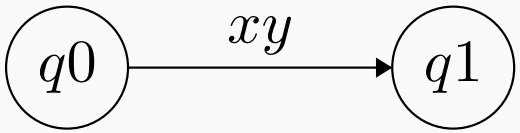
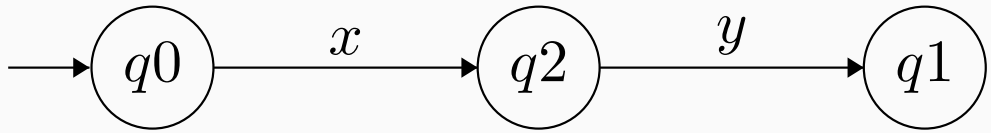
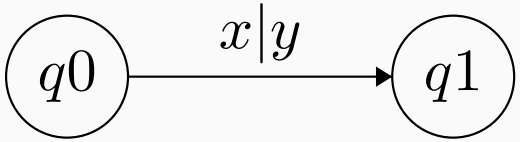
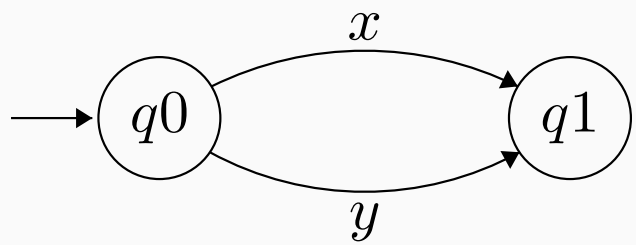
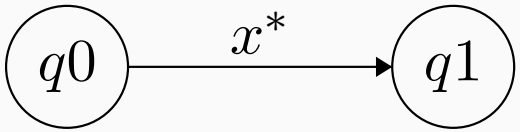
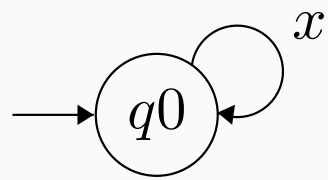
Converting REs to FSAs

Basic Recipe

Given a regular expression R:

1. Reduce the regular expression to sequence of concatenation, alternation, and Kleene star
2. Create an FSA with an initial state and a final state whose transition is R.
3. Take the lowest precedence operator in R, and deconstruct it according to the type of operator (next slide).
4. Repeat until all transitions are single elements (i.e. not REs).

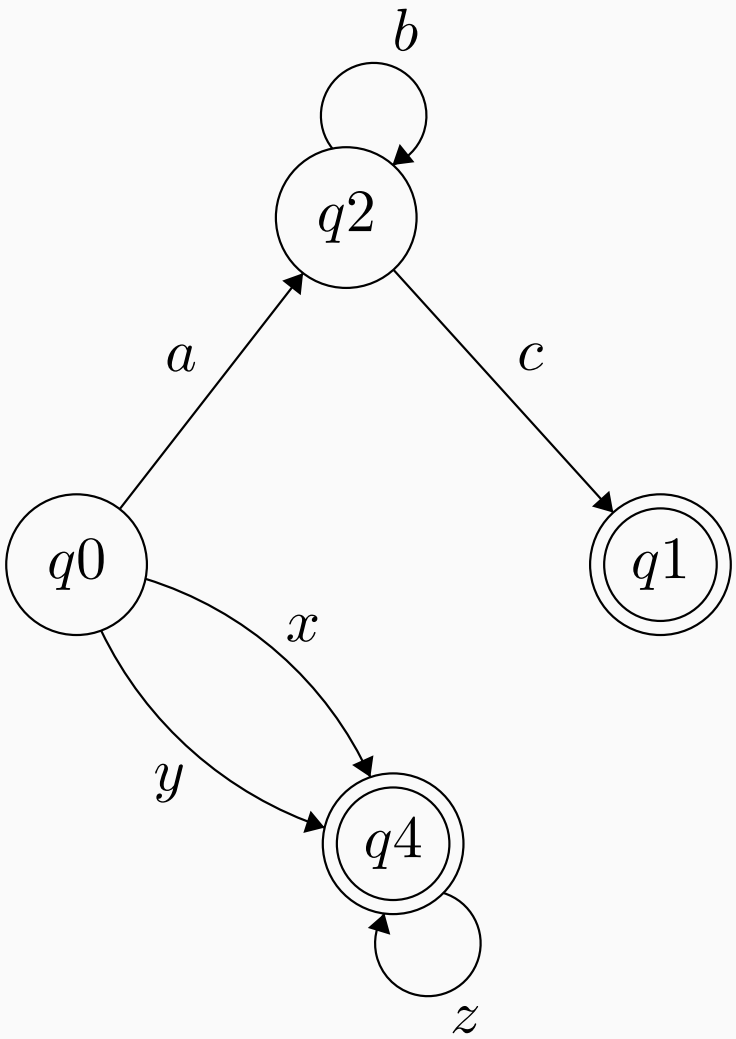
Basic operators as FSAs

Regex as state transition	Decomposed FSA
	
	
	

Question: What happens to the decomposition of x^* if $q1$ is an accept state?

Example 1

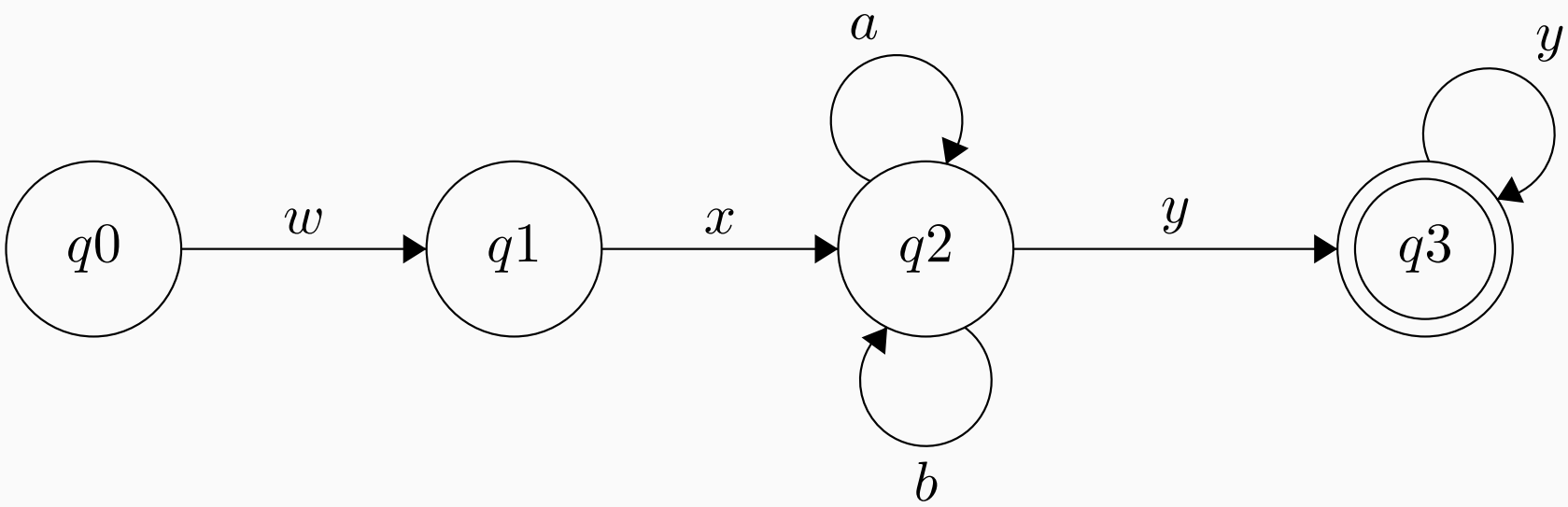
(ab*c)|(x|y)z*



Example 2

wx(a|b)*y+

Equivalent to: /wx(a|b)*yy*/



Converting REs to FSAs

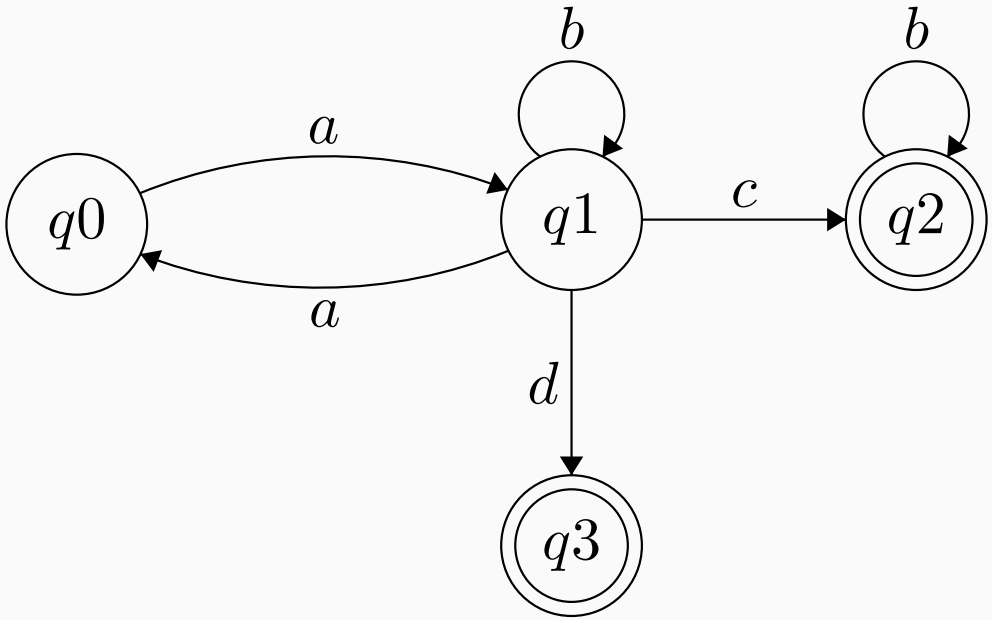
State removal method

Basic idea: Remove states in an FSA one by one and replace relevant transitions with regular expressions

Procedure, given an FSA F:

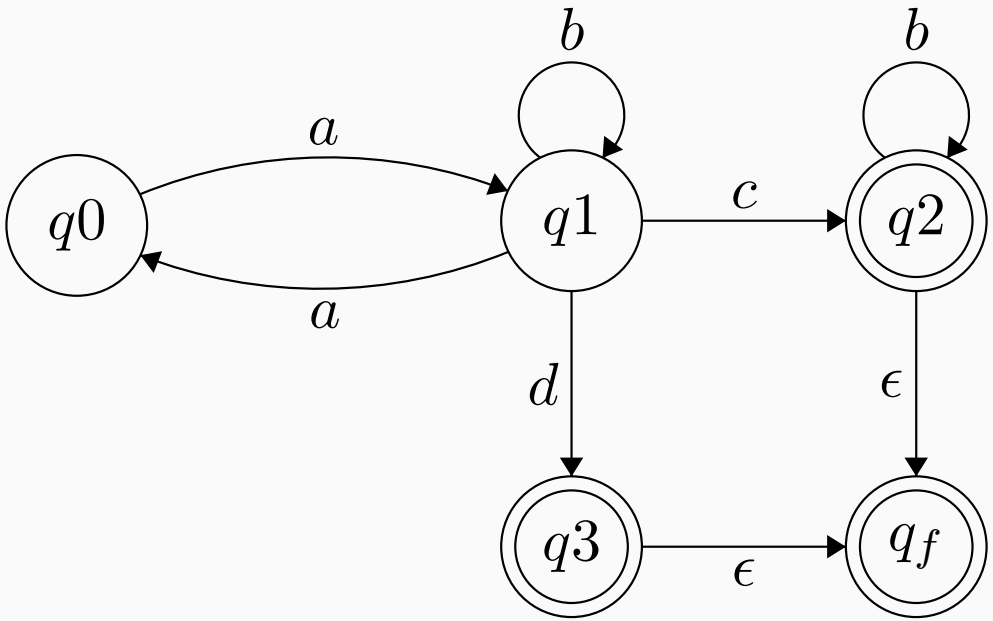
1. Add an accepting state q_f to F with ϵ-transitions from all other accepting states to q_f.
2. Turn all accepting states except q_f into non-accepting states.
3. Pick a non-accepting state q_{na} besides the start state. Replace the paths between each pair of neighbors of q_{na} with equivalent regular expressions, then delete q_{na}.
4. Repeat step 3 until only the start state and q_f remain.

Example



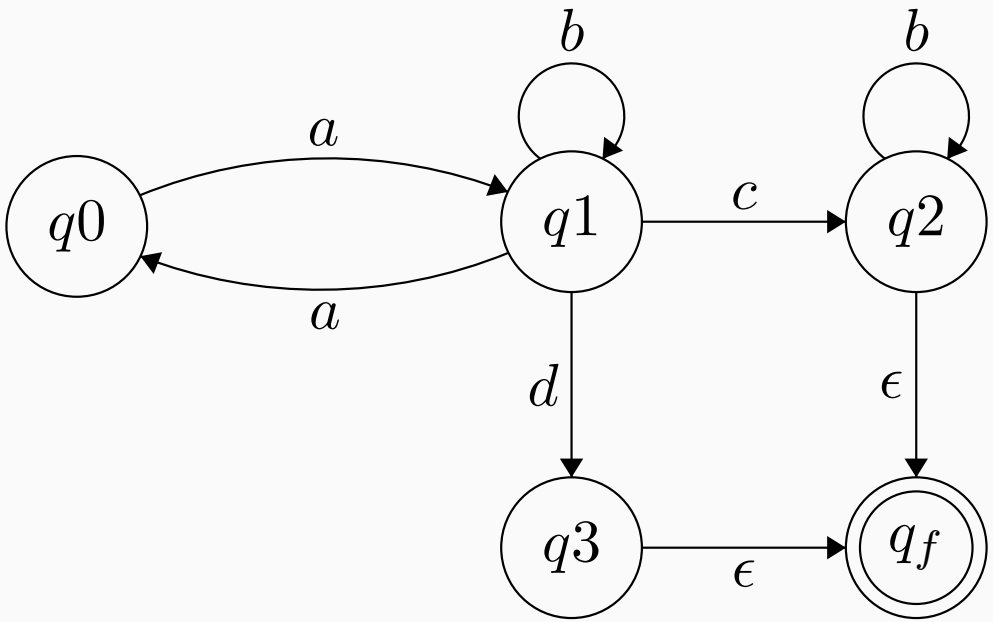
Example

Step 1: Add an accepting state q_f to F with ϵ-transitions from all other accepting states to q_f.



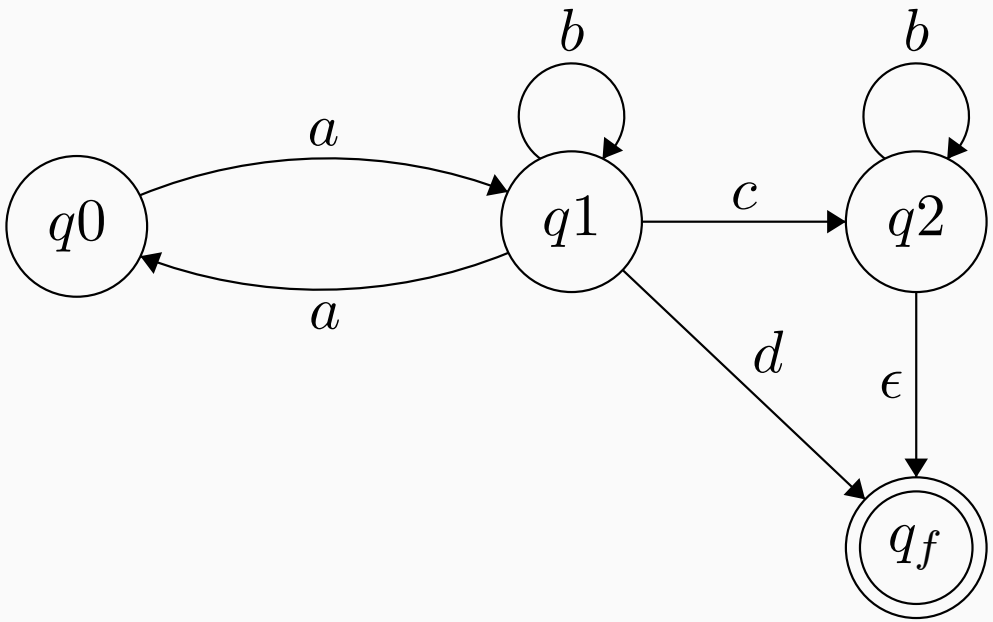
Example

Step 2: Turn all accepting states except q_f into non-accepting states.



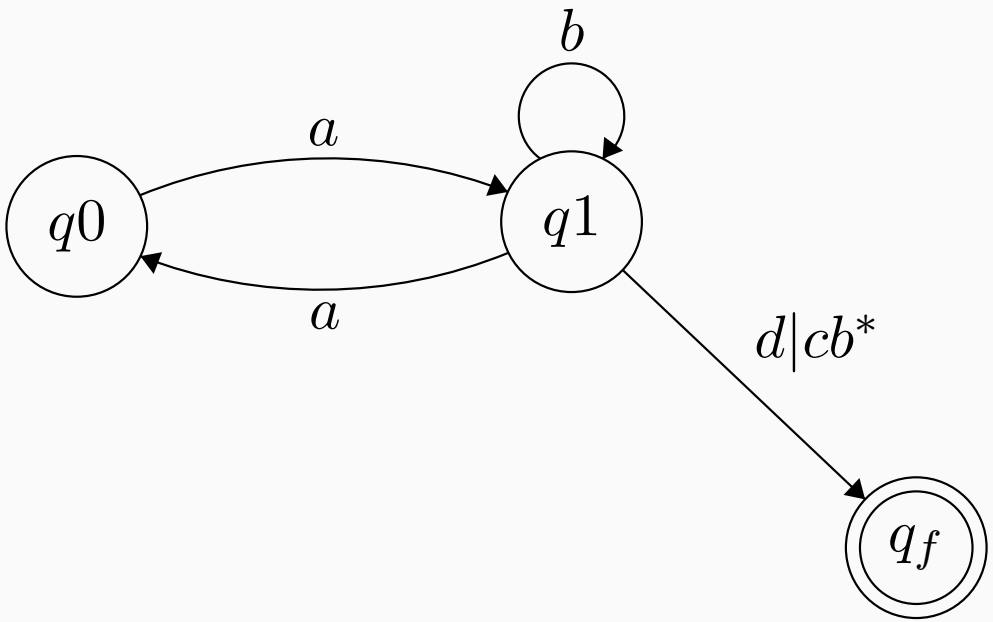
Example

Step 3: Pick a non-accepting state q_{na} besides the start state. Replace the paths between each pair of neighbors of q_{na} with equivalent regular expressions. Step 3 for q3:



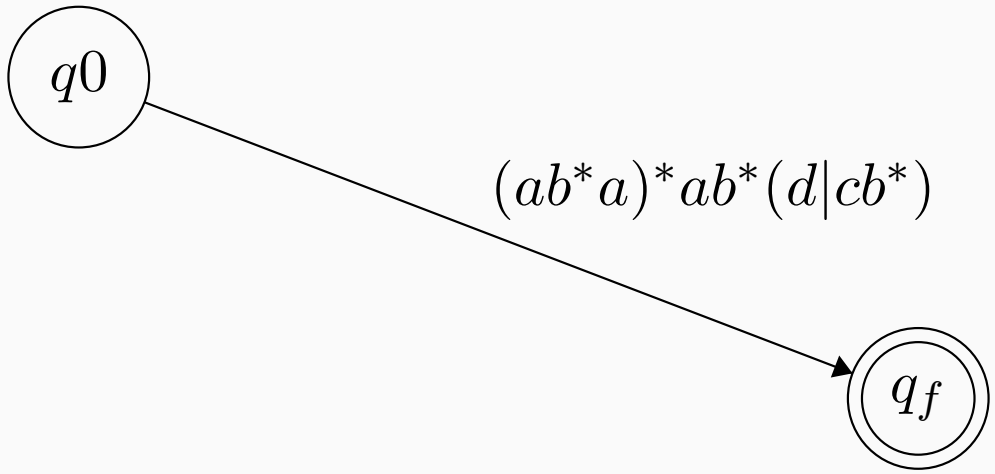
Example

Step 3: Pick a non-accepting state q_{na} besides the start state. Replace the paths between each pair of neighbors of q_{na} with equivalent regular expressions. Step 3 for q2:



Example

Step 3: Pick a non-accepting state q_{na} besides the start state. Replace the paths between each pair of neighbors of q_{na} with equivalent regular expressions. Step 3 for q1:



Double-check that this matches the original RE!

Wrapping up

- Regular languages can be represented with regular expressions or FSAs
- REs are useful for textual applications, FSAs offer a more comprehensible visualization
- Because they are equivalently expressive, we can convert between REs and FSAs